A.3 Python Special Method Reference

Adapted from https://docs.python.org/3/reference/datamodel.html#special-method-names. Note that not all special methods are shown.

A class can implement certain operations that are invoked by special syntax (such as arithmetic operations or subscripting and slicing) by defining methods with special names. This is Python's approach to operator overloading, allowing classes to define their own behavior with respect to language operators. For instance, if a class defines a method named `__getitem__()`, and `x` is an instance of this class, then `x[i]` is roughly equivalent to `type(x).__getitem__(x, i)`.

Basic customization

Method	Description
`object.__init__(self[, ...])`	Called after the instance has been created, but before it is returned to the caller. The arguments are those passed to the class constructor expression. If a base class has an `__init__()` method, the derived class's `__init__()` method, if any, must explicitly call it to ensure proper initialization of the base class part of the instance.
`object.__str__(self)`	Called by `str(object)` and the built-in functions `format()` and `print()` to compute the "informal" or nicely printable string representation of an object. The return value must be a string object.

`object.__lt__(self, other)`
`object.__le__(self, other)`
`object.__eq__(self, other)`
`object.__ne__(self, other)`
`object.__gt__(self, other)`
`object.__ge__(self, other)`
	These are the so-called "rich comparison" methods.
	The correspondence between operator symbols and method names is as follows:
	• `x < y` calls `x.__lt__(y)`
	• `x <= y` calls `x.__le__(y)`
	• `x == y` calls `x.__eq__(y)`
	• `x != y` calls `x.__ne__(y)`
	• `x > y` calls `x.__gt__(y)`
	• `x >= y` calls `x.__ge__(y)`

Emulating container types

The following methods can be defined to implement container objects. Containers usually are sequences (such as lists or tuples) or mappings (like dictionaries), but can represent other containers as well.

Method	Description
`object.__len__(self)`	Called to implement the built-in function `len()`. Should return the length of the object, an integer `>=` 0.
`object.__getitem__(self, key)`	Called to implement evaluation of `self[key]`. For sequence types, the accepted keys should be integers and slice objects. Note that the special interpretation of negative indexes (if the class wishes to emulate a sequence type) is up to the `__getitem__()` method. If key is of an inappropriate type, `TypeError` may be raised; if of a value outside the set of indexes for the sequence (after any special interpretation of negative values), `IndexError` should be raised. For mapping types, if key is missing (not in the container), `KeyError` should be raised.
`object.__setitem__(self, key, value)`	Called to implement assignment to `self[key]`. Same note as for `__getitem__()`. This should only be implemented for mappings if the objects support changes to the values for keys, or if new keys can be added, or for sequences if elements can be replaced. The same exceptions should be raised for improper key values as for the `__getitem__()` method.
`object.__contains__(self, item)`	Called to implement membership test operators (`in` and `not in`). Should return `True` if `item` is in `self`, `False` otherwise. For mapping objects, this should consider the keys of the mapping rather than the values or the key-item pairs.
`object.__iter__(self)`	This method is called when an iterator is required for a container. This method should return a new iterator object that can iterate over all the objects in the container. For mappings, it should iterate over the keys of the container.